IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

POP-UP LIGHT FIELD

Inventors:

Heung-Yeung Shum

Jian Sun

Shuntaro Yamazaki

Yin Li

ATTORNEY'S DOCKET NO. MS1-1884US

POP-UP LIGHT FIELD

TECHNICAL FIELD

[0001] The present invention generally relates to digital imaging technology. More particularly, one of the described implementations utilizes pop-up light fields of a scene to present a novel virtual view of the scene.

BACKGROUND

[0002] As computer technology improves, computer systems with more powerful processor(s) and larger storage unit(s) become more commonplace. With this growth of processing power and storage unit size, implementation of digital imaging technology also becomes more practical. For example, higher resolution images may be processed in a shorter time period.

[0003] An advantage of digital imaging technology is the ability to render the digital images. Rendering an image generally involves producing a synthetic or virtual image using a computer. For example, different light sources may be applied to a scene from different angles and with different intensities to generate a virtual view of the scene. One type of rendering is image-based rendering (IBR), where rendering techniques are applied to a set of sample input images (e.g., digital pictures taken by a digital camera or conventional pictures scanned into a computer).

[0004] Central to many IBR systems is the goal of interpolating accurately between the sample images in order to generate novel views. In IBR, rendering a desired pixel is often equivalent to interpolating intensity values of some input pixels. Such an interpolation, however, depends on the correspondence between the rendered pixel and those pixels from the input sample images. Often, accurate correspondence between these pixels can be obtained if a large number of input images or an accurate geometric model of the scene is available.

[0005] When such information is unavailable, one solution is to perform stereo reconstruction or to establish correspondence between pixels of the input images. However, state-of-the-art automatic stereo algorithms are inadequate for producing sufficiently accurate depth information for realistic rendering when using a relatively sparse set of images of a complex scene. Typically, the areas around occlusion boundaries in the scene have the least desirable results, because it is very hard for stereo algorithms to handle occlusions without prior knowledge of the scene.

[0006] Accordingly, current solutions fail to produce virtual views free of aliasing when using a relatively sparse set of images of a complex scene.

SUMMARY

[0007] Techniques are disclosed to present virtual views of a complex scene. The virtual views are substantially free from aliasing even when using a relatively sparse set of images of the scene.

[0008] In a described implementation, a scene is split into one or more coherent layers. The boundaries of the coherent layers are propagated across a plurality of frames corresponding to the scene. The splitting may be further refined (e.g., in accordance with user feedback) to present a virtual view of the scene.

[0009] In another described implementation, a user interface (UI) includes a layer pop-up module to allow a user to define one or more coherent layers corresponding to a scene. A refinement module within the UI permits the user to refine the coherent layers (e.g., to ensure substantially anti-aliased rendering). The UI further includes a rendering module to render the refined coherent layers to present a virtual view of the scene.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The detailed description is described with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items.

[0011] Fig. 1 illustrates an image rendered by applying conventional sparse light field rendering.

[0012] Fig. 2 illustrates an exemplary view of the image illustrated in Fig. 1 after application of pop-up light field rendering.

[0013] Fig. 3 illustrates a sparse light field with a single focal plane 302 in the illustrated scene (which corresponds to the image of Fig. 1).

[0014] Fig. 4 illustrates an intermediate image (between the images of Figs. 1 and 2) with two layers 402 and 404 which in combination represent the single plane 302 of Fig. 3.

[0015] Fig. 5 illustrates an exemplary pop-up light field with four layers corresponding to the image of Fig. 2.

[0016] Fig. 6 illustrates an exemplary light field 600 with coherent layers.

[0017] Fig. 7 illustrates an exemplary coherence matting method 700 which may be utilized to construct the alpha mattes in a coherent layer that have in-between frame coherence.

[0018] Fig. 8 illustrates an exemplary feathering function that defines the α value for surrounding pixels of a layer boundary.

[0019] Fig. 9 illustrates an exemplary image (which is a small window on one frame in the plaza data of Fig. 23).

[0020] Fig. 10 illustrates exemplary alpha epipolar plane images (α-EPI) corresponding to the scanline 902 of Fig. 9.

[0021] Fig. 11 illustrates exemplary alpha curves for the two adjacent lines 1006 and 1008 of Fig. 10.

[0022] Figs. 12-14 illustrate a comparison of video matting and coherence matting.

[0023] Fig. 15 illustrates an exemplary UI flow diagram for the pop-up light field construction.

[0024] Fig. 16 illustrates an exemplary editing frame view for the UI of Fig. 15.

[0025] Fig. 17 illustrates an exemplary reference view for the UI of Fig. 15.

[0026] Fig. 18 illustrates an exemplary boundary monitor showing neighboring frames of the active layer.

[0027] Fig. 19 illustrates an exemplary frame navigator for the UI of Fig. 15.

[0028] Fig. 20 illustrates an exemplary layer navigator for the UI of Fig. 15.

[0029] Fig. 21 illustrates an exemplary neighboring frame selection display for a 2D array.

[0030] Fig. 22 illustrates an exemplary background construction method 2200 which fills the disoccluded region using pixels from neighboring frames.

[0031] Fig. 23 illustrates an exemplary sample image of a plaza.

[0032] Fig. 24 illustrates an exemplary segmentation of a background layer of the image of Fig. 23 into sub layers using polygons.

[0033] Fig. 25 illustrates an exemplary resulting background corresponding to Fig. 24 wherein many missing pixels are filled.

[0034] Figs. 26-28 illustrate exemplary results when applying local geometry versus global geometry.

[0035] Fig. 29 illustrates a general computer environment 2900, which can be used to implement the techniques described herein.

[0036] Fig. 30 illustrates an exemplary block diagram that shows further details of the system memory 2906 of Fig. 29.

[0037] Fig. 31 illustrates an exemplary single pass rendering method 3100 to provide improved speed.

DETAILED DESCRIPTION

[0038] The following disclosure describes the generation of novel virtual views of a scene that are substantially free from aliasing even when using a relatively sparse set of images of a complex scene. An interactive image-based modeling and rendering solution is provided to limit the anti-aliasing affects. More specifically, multiple images of a same subject are taken from different angles. A user may then interactively cut out (or define) different objects (including backgrounds) within the images. The cut-outs may be treated as different coherent layers. Using the multiple images and the layers, different photorealistic views of the subject may be rendered. Coherence matting (which uses Bayesian techniques) can be used to reduce or eliminate problems present at the boundaries of the defined objects.

[0039] **ANTI-ALIASING EXAMPLES**

[0040] Fig. 1 illustrates an image rendered by applying conventional sparse light field rendering. As can be seen, objects within the illustrated light field of Fig. 1 are aliased. Aliasing is especially visible near front objects (such as the lamp) because the input light field is sparse.

[0041] Fig. 2 illustrates an exemplary view of the image illustrated in Fig. 1 after application of pop-up light field rendering. As illustrated, the anti-aliased rendering can be achieved when applying the techniques discussed herein. For

example, the front objects (such as the lamp) are substantially anti-aliased when compared with the front objects of Fig. 1.

[0042] **PROGRESSION OF POP-UP LIGHT FIELD RENDERING**

[0043] Figs. 3-5 illustrate exemplary progression of pop-up light field rendering corresponding with the images of Figs. 1 and 2. In particular, Fig. 3 illustrates a sparse light field with a single focal plane 302 in the illustrated scene (which corresponds to the image of Fig. 1). Fig. 4 illustrates an intermediate image (between the images of Figs. 1 and 2) with two layers 402 and 404 which in combination represent the single plane 302 of Fig. 3. Fig. 5 illustrates an exemplary pop-up light field with four layers (402, 502, 504, and 506) corresponding to the image of Fig. 2. As illustrated, objects within each of the four layers (402, 502, 504, and 506) are substantially anti-aliased in Fig. 2.

[0044] Accordingly, by splitting the scene into multiple layers, the depth variation in each layer becomes much smaller than that in the original sparse light field. Also, the pop-up light field is represented by a collection of coherent layers (as will be discussed further with respect to the remaining figures such as Fig. 6). In an implementation, the number of coherent layers that should be modeled or "popped up" depends on the complexity of the scene and/or how under-sampled the input light field is. For example, for a sparser light field, more layers may be popped up for substantially anti-aliased rendering.

[0045] **COHERENT LAYERS**

[0046] Fig. 6 illustrates an exemplary light field 600 with coherent layers. The light field 600 includes images 602 (I_1) and 604 (I_2) that are represented by a set of coherent layers 606 (L_1) and 608 (L_2), respectively. Generally, a coherent layer is a collection of layered images in the light field. For instance, the layer 606 (L_1) is represented by layered images 610 (from I_1) and 612 (from I_2). Similarly, the layer 608 (L_2) is represented by layered images 614 (from I_1) and 616 (from I_2). Part of the scene corresponding to each layer is, in turn, modeled as a plane (e.g., P_1 for L_1 and P_2 for L_2, respectively).

[0047] As illustrated in Fig. 6, a coherent layer L_j can be represented by a collection of corresponding layered image regions R_j^i in the light field images I^i. These regions can be modeled by a simple geometric proxy without the need for accurate per-pixel depth. For example, a global planar surface (P_j) is used as the geometric proxy for each layer L_j in the example shown in Fig. 6. To deal with complicated scenes and camera motions, a local planar surface P_j^i may be utilized to model the layer in every image i of the light field.

[0048] A layer in the pop-up light field is considered as "coherent" if the layer can be rendered substantially free of aliasing by using a simple planar geometric proxy (global or local). Anti-aliased rendering may occur at two levels, when: (1) the layer itself is rendered; and (2) the layer is rendered with its background layers. Therefore, to satisfy the first condition, the depth variation in

each layer can be configured to be sufficiently small. Moreover, the planar surface can be adjusted interactively to achieve the best rendering effect.

[0049] To meet the second condition, accurate layer boundaries can be maintained across the frames to construct the coherent layers. In one implementation, to ensure segmentation coherence across a group of the frames, the segmented regions on one or more key frames are propagated to the remaining frames. Sub-pixel precision segmentation may be obtained on the key frames by meticulously zooming on the images and tracing the boundaries. Propagation from key frames to other frames, however, may cause under-segmentation or over-segmentation of a foreground layer. Typically, over-segmentation of a foreground layer leads to the inclusion of background pixels; thus, introducing ghosting along the occlusion boundaries in the rendered image. To alleviate the rendering artifacts caused by over-segmentation or under-segmentation of layers, the layer boundary may be refined with alpha matting. For example, with respect to Fig. 6, the pixels at each coherent layer have consistent depth values (e.g., within a depth bound), but may have different fractional alpha values along the boundary. Accordingly, each of the layered images may have an alpha matte associated with its boundary.

[0050] To produce fractional alpha mattes for the regions in a coherent layer, video matting may be applied. The video matting problem is formulated as a maximum a posterior (MAP) estimation as in Bayesian matting:

$$\arg\max_{F,B,\alpha} P(F,B,\alpha|C)$$
$$= \arg\max_{F,B,\alpha} L(C|F,B,\alpha) + L(F) + L(B) + L(\alpha) \quad (1)$$

where $L(\bullet) = logP(\bullet)$ is log likelihood, C is the observed color for a pixel, and F, B and α are foreground color, background color, and alpha value to be estimated, respectively. For color images, C, F, and B are vectors in red-green-blue (RGB) space. In Bayesian matting and video matting, the log likelihood for the alpha $L(\alpha)$ is assumed to be constant, so that $L(\alpha)$ is dropped from Equation (1).

[0051] In video matting, the optical flow is applied to the tri-map (the map of foreground, background, and uncertain region), but not to the output matte. The output foreground matte is produced by Bayesian matting on the current frame, based on the propagated tri-map. In an implementation, to make video matting work well, the foreground mattes are replayed against a different background. However, these foreground mattes may not have in-between frame coherence that may be needed for generating novel views.

[0052] **COHERENCE MATTING**

[0053] Fig. 7 illustrates an exemplary coherence matting method 700 which may be utilized to construct the alpha mattes in a coherent layer that have in-between frame coherence. In one implementation, the method 700 may have a workflow similar to video matting. In a stage 702, segmentation of the scene is determined. For example, a user may specify approximate boundaries for select

objects (e.g., the layer 402 of Fig. 4). A stage 704 propagates the segmentation boundaries across frames. In a stage 706, an uncertain region along the boundary is determined. As a result, the uncertain region is between the foreground and background. A stage 708 combines the under-segmented background regions from multiple images to construct a sufficient background image. Alpha matte for the foreground image (i.e., in the uncertain region) is estimated in a stage 710.

[0054] The method 700 may also include an optional stage 712 to enable refinement of the foreground (e.g., through user feedback). A stage 714 constructs the coherent foreground layer. In an implementation, a coherent feathering function (such as the one illustrated in Fig. 8 and further discussed below) is applied across the corresponding layer boundaries. Also, for a given layer, a separate foreground matte may be estimated independently for each frame in the light field, and the coherence across frames may be maintained by foreground boundary consistency.

[0055] *L(B)* in Equation (1) can be dropped since the background is explicitly estimated. By incorporating a coherence prior on the alpha channel *L(α)* across frames, coherence matting can be formulated as:

$$L(F,B,\alpha|C) = L(C|F,B,\alpha) + L(F) + L(\alpha) \qquad (2)$$

where the log likelihood for the alpha *L(α) is modeled as:*

$$L(\alpha) = -\frac{(\alpha-\alpha_0)^2}{\sigma_a^2} \qquad (3)$$

where $\alpha_0 = f(d)$ is a feathering function of d, σ_a^2 is the standard deviation, and d is the distance from the pixel to the layer boundary.

[0056] The feathering function $f(d)$ defines the α value for surrounding pixels of a boundary. In an implementation, the feathering function is set as: $f(d)=(d/w)*0.5+0.5$, where w is feathering width, as illustrated in Fig. 8.

[0057] Assuming that the observed color distribution $P(C)$ and sampled foreground color distribution $P(F)$ (from a set of neighboring foreground pixels) are of Gaussian distribution:

$$L(C|F,\mathrm{B},\alpha) = -\| |C-\alpha F-(1-\alpha)B| \|^2/\sigma_C^2 \tag{4}$$

$$L(F) = -(F-\bar{F})^T \sum_F^{-1} (F-\bar{F}) \tag{5}$$

where σ_C is the standard deviation of the observed color C, $\bar{F}$ is the weighted average of foreground pixels, and $\sum_F$ is the weighted covariance matrix. Taking the partial derivatives of equation (2) with respect to F and α and forcing them equal to zero results in the following equations:

$$F = \frac{\sum_F^{-1} \bar{F} + C\alpha/\sigma_C^2 - B\alpha(1-\alpha)/\sigma_C^2}{\sum_F^{-1} + I\alpha^2/\sigma_C^2} \tag{6}$$

$$\alpha = \frac{(C-B)\cdot(F-B) + \alpha_0 \cdot \sigma_C^2/\sigma_a^2}{\| |F-B| \|^2 + \sigma_C^2/\sigma_a^2} \tag{7}$$

where α and F are solved alternatively by using equations (6) and (7). Initially, α is set to α_0 through a curve editing tool because the width and

shape of feathering function depend on the resolution of the image, image sensor (e.g., the sensor point spread function), and the scene.

[0058] Furthermore, Bayesian matting and video matting solve the matting from the equation:

$$\alpha = \frac{(C-B)\cdot(F-B)}{\left|\,|F-B|\,\right|^2} \tag{8}$$

[0059] Equation (8) works well in general but becomes unstable when $F \approx B$. In comparison, the coherence matting of Equation (7) can be solved more stably, because applying the coherence prior on α results in a non-zero denominator.

[0060] **VIDEO MATTING VERSUS COHERENCE MATTING**

[0061] Fig. 9 illustrates an exemplary image 900 (which is a small window on one frame in the plaza data of Fig. 23). The image of Fig. 9 includes a redline 902 which will be utilized as a reference point when discussing Figs. 10 and 11.

[0062] Fig. 10 illustrates exemplary alpha epipolar plane images (α-EPI) 1002 and 1004 corresponding to the scanline 902 of Fig. 9, using the algorithm of video matting and coherence matting respectively. The alpha values (ranging between 0 and 255) along the scanline 902 are presented for the available 16 frames (i) in images 1002 and 1004. Fig. 10 also includes a solid line 1006 and a dotted line 1008 for reference purposes, which correspond to pixels across the 16 frames. A large jump is clearly visible at 1010 (e.g., where $i=13$ in the video matting approach).

[0063] Fig. 11 illustrates exemplary alpha curves for the two adjacent lines 1006 and 1008 of Fig. 10. In particular, alpha curves 1102 and 1104 correspond to the images 1002 and 1004, respectively. A curve 1106 corresponds to the line 1006 of image 1002 and a curve 1108 corresponds to the line 1008 of image 1002. Similarly, a curve 1110 corresponds to the line 1006 of image 1004 and a curve 1112 corresponds to the line 1008 of image 1004.

[0064] As can be seen, coherence matting (e.g., of image 1004 and alpha curve 1104) provides a more reasonable result when compared with video matting (e.g., of image 1002 and alpha curve 1102). For example, the jump at 1010 causes an accidental transparency within the face illustrated in Fig. 9, because the alpha value changes from about 126 to 0 and then to 180.

[0065] The temporal incoherence of the alpha matte from video matting can be more problematic during rendering. The fluctuation of alpha values along both dotted and solid lines (1106 and 1108) can generate incoherent alpha values and thus cause rendering artifacts for different viewpoints (i.e., along axis *i*). The alpha curve 1104 shows the same solid and dotted lines (1110 and 1112) with coherent matting results. Both lines have much less fluctuation between neighboring pixels, and appear temporally smoother than their counterparts in the alpha curve 1102.

[0066] A further comparison of video matting and coherence matting are illustrated with reference to Figs. 12-14. In particular, Fig. 12 illustrates an exemplary reference image. The image of Fig. 12 includes a portion 1202

(corresponding to an ear of a furry rabbit). Figs. 13 and 14 illustrate exemplary results of applying video matting (Fig. 13) and coherence matting (Fig. 14) to the portion 1202 of Fig. 12. As shown, the alpha matte from coherent matting (Fig. 14) is smoother that that from video matting (Fig. 13) in the rendered image.

[0067] **USER INTERFACE FOR POP-UP LIGHT FIELD CONSTRUCTION**

[0068] To construct a pop-up light field, a UI may be utilized. Through the UI, a user may specify, refine, and propagate layer boundaries, and indicate rendering artifacts. More layers may be popped up and refined until the user is satisfied with the rendering quality.

[0069] Fig. 15 illustrates an exemplary UI flow diagram for the pop-up light field construction. A sparse light field repository 1502 is accessed by a layer pop-up module 1504. The layer pop-up module 1504 receives instructions from a user input module 1506. A background construction module 1508 receives layer pop-up information from the layer pop-up module and user input from the user input module 1506 to construct the background layer. A foreground refinement module receives background construction information from the module 1508 and user input from the module 1506 to refine the foreground layer. The output of the foreground refinement module 1510 is stored in pop-up light field repository 1512 and then provided to a pop-up light field rendering module 1514. The user may receive information from the pop-up light field rendering module 1514 and utilize

it to provide appropriate input to the modules 1504, 1508, and 1510 to improve the quality of the rendered image.

[0070] Accordingly, a user can supply the information needed for layer pop-up (1504), background construction (1508), and foreground refinement (1510). By visually inspecting the rendering image from the pop-up light field (1514), the user may also indicate where aliasing occurs and thus which layer needs to be further refined. The user input or feedback may be automatically propagated across a group of the frames in the pop-up light field. The four stages of operations in the UI discussed with reference to Fig. 15 may be further summarized as follows:

[0071] (1) Layer Pop-Up (1504). This stage segments layers and specifies their geometries. To start, the user selects a key frame in the input light field, specifies regions that need to be popped up, and assigns the layer's geometry by either a constant depth or a plane equation. This stage results in a coarse segmentation represented by a polygon. The polygon region and geometry configuration can be automatically propagated across frames. Layers should be popped up in order of front-to-back in an implementation.

[0072] (2) Background Construction (1508). This stage obtains background mosaics that are needed to estimate the alpha mattes of foreground layers. Note that the background mosaic is useful only for the pixels around the foreground boundaries, i.e., in the uncertain region as discussed with reference to

Fig. 7. Further details regarding background construction will be discussed with reference to Fig. 22.

[0073] (3) Foreground Refinement (1510). Based on the constructed background layers (1508), this stage refines the alpha matte of the foreground layer by applying the coherence matting algorithm. Unlike layer pop-up in stage (1), foreground refinement in this stage should be performed in back-to-front order in one implementation.

[0074] (4) Rendering Feedback (1512 and 1514). Any modification to the above stages (e.g., through 1506) will update the underlying pop-up light field data. The rendering window will be refreshed with the changes as well. By continuously changing the viewpoint the user can inspect for rendering artifacts. The user can mark any rendering artifacts such as ghosting areas by brushing directly on the rendering window. The corresponding frame and layer will then be selected for further refinement.

[0075] **USER INTERFACE DESIGN**

[0076] Figs. 16-20 illustrate exemplary workspaces associated with the UI discussed with reference to Fig. 15, through which the user interacts with a frame and/or a layer in the pop-up light field.

[0077] Fig. 16 illustrates an exemplary editing frame view for the UI of Fig. 15. The user can create or select an active layer and edit its polygon region. This

active layer is displayed in polygons with crosses for each editable vertex (e.g., 1602 and 1604, respectively). The information regarding the active layer may be available in the layer navigator (shown at in Fig. 20).

[0078] Fig. 17 illustrates an exemplary reference view for the UI of Fig. 15. The reference frame view may be used to display a different frame (than the one shown in Fig. 16 for example) in the light field. This workspace is useful for a number of operations where correspondences between the reference frame view (Fig. 17) and the editing frame view (Fig. 16) need to be considered, such as specifying plane equations. The reference view of Fig. 17 may also display polygons with crosses for each editable vertex (e.g., 1702 and 1704, respectively).

[0079] Fig. 18 illustrates an exemplary boundary monitor showing neighboring frames (1802) of the active layer (such as that shown in Fig. 16). Fig. 18 shows close-up views of multiple frames in the light field. Fig. 18 may be utilized to fine-tune the polygon location for the active layer. A key point marker 1804 is shown for each frame. The active layer may appear in the middle (1806) of the neighboring frames (1802). The first row (1808) shows the close-up around the moving vertex. The second (1810) and third (1812) rows show the foreground and background of the active layer composed with a fixed background selected by the user. For instance, using mono fuchsia color in Fig. 18 as the background, it is easy for the user to observe over-segmentation or under-segmentation of the foreground across multiple frames simultaneously.

[0080] Fig. 19 illustrates an exemplary frame navigator for the UI of Fig. 15. The user may choose an active frame by clicking on a location marker (1902) of the frame navigator. One or more key points (1904) may be highlighted in the frame navigator with distinguishing indicia (such as a circle). The key points may correspond with the key point markers 1804 of Fig. 18.

[0081] Fig. 20 illustrates an exemplary layer navigator for the UI of Fig. 15. The user may obtain the active layer's information in Fig. 20. By utilizing a list 2002 (e.g., by selecting a layer which may be illustrated by a check mark in the list 2002), a user may view (2004), add (2006), or delete (2008) layers. For example, by selecting the layer in the check box of the list 2002, the user can turn on/off a layer's display in the editing frame view (Fig. 16) and reference frame view (Fig. 17). The plane equation of the active layer can be displayed and modified through user input (2010, 2012, and 2014).

[0082] In one implementation, layer equations can also be set through adjusting the rendering quality in a rendering window (not shown). A rendering window may enable display of any novel view in real time and allow the user to inspect the rendering quality. The user may also specify the frontal plane's equation for an active layer by sliding a plane depth back and forth until the best rendering quality (i.e., minimum ghosting) is achieved. If the ghosting cannot be completely eliminated at the occlusion boundaries, the layer's polygon may be fine tuned (e.g., by utilizing the editing view of Fig. 16). The user can brush on the

ghosting regions, and the system can automatically select the affected frame and layer for modification. In one implementation, the affected layer is the front-most and closest to the specified ghosting region.

[0083] To specify the slant plane equation for a layer (2014), the user may select at least four pairs of corresponding points on the editing frame view (Fig. 16) and the reference frame view (Fig. 17). The plane equation can be automatically computed and then used for rendering.

[0084] In another implementation, the user can specify the feathering function (such as discussed with reference to Fig. 8). Specifying a feathering curve may be useful for the coherence matting algorithm discussed with reference to Fig. 7.

[0085] **LAYER INITIALIZATION AND REFINEMENT**

[0086] To pop up a layer, the user segments and specifies the geometry of the active layer for a group of frames in the light field. To initialize a layer, polygons are used to represent layer boundaries (in part, because the correspondence between polygons can be maintained well in the frames by the corresponding vertices). The user can specify the layer's boundary with a polygon (e.g., using the polygon lasso tool and edit the polygon by dragging the vertices). The editing may be immediately reflected in the boundary monitor window such as Fig. 18 (and/or in the rendering window). The user may then select a proper key frame to work with and draws a polygon on the frame.

[0087] To specify the layer's geometry, the user provides further input. For a frontal plane, the layer depth is used to achieve the best rendering quality (e.g., as observed by the user). For a slant plane, the user specifies at least four pairs of corresponding points on at least two frames to estimate the plane equation.

[0088] Once the layer geometry is decided, the polygon on the first key frame can be propagated to the other frames by back projecting its vertices, resulting in a coarse segmentation of the layer on the frames in the light field. All vertices on the key frame may be marked as key points. At this stage, the layer has a global geometry which is shared across the frames.

[0089] Moreover, an accurate polygon boundary for layer initialization is not necessary. Because of occlusions and viewpoint changes, propagated polygon boundaries may need to be refined. More specifically, boundary refinement in a key frame may be achieved by adding, deleting, and moving the vertices on any frame. Once a vertex is modified, it may be marked as a key point. The position of the modified vertex can be propagated across frames and the layer region will be updated in several UI workspaces. To adjust a vertex position, the user can observe how well foreground and background colors are separated in the boundary monitor window (Fig. 18), or how much the ghosting effect is removed in the rendering window.

[0090] To provide boundary propagation across multiple frames, for a specific vertex on the layer boundary, if there is a non-key point on frame I_P, its

image coordinate from the corresponding key points in other frames are interpolated. If there is only one key point in other frames, the coordinate may be computed by back projecting the intersection point of layer plane and the viewing ray from the key point. Otherwise, two or three "neighboring" frames may be selected that contain the key points. Then, the coordinate may be calculated by back projecting the 3D point which has minimal sum of distances to the viewing rays from key points in the frames under consideration.

[0091] For a 1D camera array, the two frames closest to the left and right of the frame I_P may be selected (that contain key points). For a 2D camera array, the Delaunay triangulation in the camera plane may be calculated by using all frames containing key points. Generally, Delaunay triangulation involves drawing a line between any two points whose Voronio domains touch to obtain a set of triangles. Given a set of points (such as key points discussed herein), obtaining Voronio domains involves splitting a plane in domains for which the first point is closest, the second point is closest, etc. Accordingly, if a frame I_P is on the interior of a triangle, the three frames on the triangle vertices are considered "neighboring" frames.

[0092] Fig. 21 illustrates an exemplary neighboring frame selection display for a 2D array. In Fig. 21, A, B, and D (2102) are the "neighboring" frames of frame b (2104). If frame I_P is in the exterior of all triangles, two frames I_0 and I_1 may be selected that maximize the angle $\angle I_0 I_P I_1$ in camera plane. For example, A

and *D* are the "neighboring" frames of a frame *a* (2106), as shown in Fig. 21. In one implementation, key points are those that have been modified by the user. They don't necessarily exist on key frames.

[0093] Coherence matting (which was discussed with respect to Fig. 7) may be utilized to assist a user in describing a layer boundary more accurately than just using polygons. For example, it cumbersome for the user to manually adjust to sub pixel accuracy a boundary with subtle micro geometry. A pixel is often blended with colors from both foreground and background due to the camera's point spread function. Therefore, the user is not required to specify very accurate sub-pixel boundary positions. Instead, the coherence matting algorithm (Fig. 7) may be applied to further refine the layer boundary. Polygon editing (in a frame and across frames) and coherence matting can be alternatively performed with assistance from the user (see, e.g., Fig. 15).

[0094] **CONSTRUCTING THE BACKGROUND**

[0095] The coherence matting method discussed with reference to Fig. 7 assumes that the background for the uncertain regions (where matting is estimated) is known. Since the uncertain regions are located around the foreground boundaries, they can only appear on neighboring frames in the light field where these regions are disoccluded.

[0096] Fig. 22 illustrates an exemplary background construction method 2200 which fills the disoccluded region using (warped) pixels from neighboring

frames. The method 2200 includes a stage 2202 which determines the background layer. In one implementation, once the foreground is popped up, the background image can be obtained by removing the foreground image. Moreover, the background boundary is eroded (stage 2204) by a few pixels (e.g., two pixels) before constructing the background mosaic, because a possible under-segmentation of the foreground may leave some mixed foreground pixels on the background around boundaries.

[0097] The background may be constructed by warping the neighboring images to fill the holes using the background layer's geometry. This works well if the background is well approximated by plane, e.g. in Fig. 1. Accordingly, a stage 2206 determines whether the background includes objects with large depth variation. If the stage 2206 determines that the background includes objects with large depth variation, a stage 2208 subdivides the background layer into sub layers, each of which can be represented as one plane. If the stage 2206 returns a negative response or after the stage 2208, a stage 2210 determines the disoccluded regions. The neighboring frames of the disoccluded regions are determined by a stage 2212. And, a stage 2214 fills the disoccluded regions with corresponding pixels.

[0098] The method 2200 is further discussed with reference to Figs. 23-25. Fig. 23 illustrates an exemplary sample image of a plaza. Fig. 24 illustrates an exemplary segmentation of a background layer of the image of Fig. 23 into four

sub layers (or regions) using polygons 2402, 2404, 2406, and 2408. Fig. 25 illustrates an exemplary resulting background corresponding to Fig. 24 wherein many missing pixels are filled.

[0099] As shown in Fig. 23, a background layer is segmented manually into four sub layers using polygons. It is envisioned that the location of the polygons are not critical. Instead, the goal is to group the background boundaries into a better planar approximation. The sub layers (such as those discussed with reference to Fig. 22) are propagated from the key frame, where the user specifies the division, to the other frames using the existing background layer geometry. This propagation requires less accuracy as long as it covers the same group of boundaries.

[00100] In one implementation, the relative motion of the sub layer across frames is estimated hierarchically, starting from translation to affine, and from affine to perspective transform. Only the pixels visible in both frames are used to estimate parameters. Fig. 25 shows the resulting mosaic. It is envisioned that a hole-free mosaic is unnecessary, as a few pixels surrounding the occlusion boundaries are adequate for coherence matting.

[00101] **LOCAL GEOMETRY**

[00102] When the viewpoint changes significantly, a single planar geometry may not be sufficient to achieve anti-aliased rendering. Therefore, a local

geometry representation may be utilized which allows each frame to have its own planar equation.

[00103] Using the same UI discussed with reference to Figs. 16-20, the plane equation can be estimated for each frame. The user may specify a few key frames' geometry, and the plane equations for frames in between can be interpolated. Similar to the "neighboring" frames selection in the boundary propagation, two (for 1D camera array) or three (for 2D camera array) key frames may be selected for interpolation. For the frontal plane model, the depth of the plane is interpolated. For 3D plane model, the plane orientation can be interpolated while keeping the intersecting line if using two key frames, or the intersecting point if using three key frames. Once the plane equation is estimated for each frame, the same rendering algorithm can be applied as in using a global geometry.

[00104] Figs. 26-28 illustrate exemplary results when applying local geometry versus global geometry. Fig. 26 illustrates an exemplary image with a global planar surface set as a frontal, parallel plane. Fig. 27 illustrates an exemplary rendering result from another view (other than the view shown in Fig. 26) with the global plane. Fig. 28 illustrates an exemplary view rendered with local geometry applied to Fig. 27. As can be seen, application of local geometry results in substantially more anti-aliasing, when the viewpoint changes significantly.

[00105] **REAL-TIME RENDERING OF POP-UP FIELD**

[00106] An integral part of the UI is the ability to render pop-up light field in real-time (such as the pop-up light field rendering module 1514 of Fig. 15). This provides the user with instant feedback on the rendering quality. The rendering algorithm includes three stages: (1) splitting a light field into layers, (2) rendering layers in back-to-front order, and (3) combining the layers.

[00107] The following data structure may be used to provide the rendering:

```
struct PopupLightField {
        Array<CameraParameter> cameras;
        Array<Layer> layers;
};
struct Layer {
        Array<Plane> equations;
        Array<Image> images;
};
struct Image {
        BoundingBox box;
        Array2D<RGBA> pixels;
};
```

[00108] In one implementation, the pop-up light field keeps the camera parameters associated with all the input frames. Each layer in the pop-up light

field has corresponding layered images, one for each frame. Each layered image has a corresponding plane equation, so as to represent the local geometry. If global geometry is applied to a layer, all equations are the same for images in this layer.

[00109] Since these corresponding layered images vary their shapes in different views, they are stored as an array of images on each layer. Layers can be overlapping in the pop-up light field and each layered image is modified independently by mosaicing and/or coherent matting.

[00110] Therefore, both color and opacity of images for each layer may be stored separately. In an implementation, each layered image is stored as an red-green-blue alpha (RGBA) texture image of the foreground colors with its opacity map, and a bounding box. The opacity (or alpha value) of the pixel is zero when this pixel is out of the foreground.

[00111] **LAYERED RENDERING ALGORITHM**

[00112] In an implementation, the scene is rendered layer by layer using texture-mapped triangles in back-to-front order. Then, the layers are sequentially combined by alpha blending which is extended to multiple layers. A pseudo code for the rendering is shown below:

```
ClearFrameBuffer()
T ← CreateRenderingPrimitives()
for all layers Layer from back to front do
    for all triangles Δ ∈ T do
```

SetupProjectiveTextureMapping(Δ)

Render(Δ)

BlendToFrameBuffer(Δ)

end for

end for

[00113] Accordingly, after initializing a frame buffer, a set of triangular polygons is generated on which the original images are blended and drawn. First the camera positions are projected onto the image plane and these projection points are triangulated together with the image plane's four corner points into a set of triangles.

[00114] A triple of texture images $\{I_i\}_{i=1}^{3}$ are assigned to each triangle, which are blended across the triangle when rendering. The blending ratio $\{w_i^k\}_{k=1}^{3} (0 \le w_i^k \le 1, \sum_{k=1}^{3} w_i^k = 1)$ for three images are also assigned to each of the three vertices, and linearly interpolated across the triangle. The exact blending ratio based on ray angles is not necessarily distributed linearly on the screen. If the size of a triangle is not small enough with respect to the screen size, the triangle may be subdivided into four triangles iteratively. On the vertex which is the projection of a I_i's camera, the blending ratio w_i^k is calculated using the following equation:

w_i^k =1, if camera i is projected onto the k-th vertex

= 0, otherwise

[00115] For the vertex which is not the projection of a camera, the weights can be calculated using the angle between the ray through the camera and the ray through the vertex. Then, each layer can be rendered by blending texture images $\{I_i\}$ using blending ratios $\{w_i^k\}$. At the point other than the vertices on the triangle, the blending ratios $\{\tilde{v}_i\}$ are calculated by interpolating $\{w_i^k\}_{k=1}^3$. Using $\{I_i\}$ and $\{\tilde{v}_i\}$, the pixels on the triangle are drawn in the color $\sum_{i=1}^3 \tilde{v}_i I_i$.

[00116] The texture images are mapped onto each triangle projectively. If P_{view} is the projection matrix for the rendering camera (i.e., to produce the novel view), P_i is the projection matrix for the camera corresponding to I_i, and H_{layer} be a planar homography from the triangle to the layer plane. Then, the texture image I_i can be mapped onto the triangle using a projection matrix $P_i H_{layer}$.

[00117] **HARDWARE IMPLEMENTATION**

[00118] Fig. 29 illustrates a general computer environment 2900, which can be used to implement the techniques described herein. The computer environment 2900 is only one example of a computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the computer and network architectures. Neither should the computer environment 2900 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary computer environment 2900.

[00119] Computer environment 2900 includes a general-purpose computing device in the form of a computer 2902. The components of computer 2902 can include, but are not limited to, one or more processors or processing units 2904 (optionally including a cryptographic processor or co-processor), a system memory 2906, and a system bus 2908 that couples various system components including the processor 2904 to the system memory 2906.

[00120] The system bus 2908 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, such architectures can include an Industry Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA) local bus, and a Peripheral Component Interconnects (PCI) bus also known as a Mezzanine bus.

[00121] Computer 2902 typically includes a variety of computer-readable media. Such media can be any available media that is accessible by computer 2902 and includes both volatile and non-volatile media, removable and non-removable media.

[00122] The system memory 2906 includes computer-readable media in the form of volatile memory, such as random access memory (RAM) 2910, and/or non-volatile memory, such as read only memory (ROM) 2912. A basic

input/output system (BIOS) 2914, containing the basic routines that help to transfer information between elements within computer 2902, such as during start-up, is stored in ROM 2912. RAM 2910 typically contains data and/or program modules that are immediately accessible to and/or presently operated on by the processing unit 2904.

[00123] Computer 2902 may also include other removable/non-removable, volatile/non-volatile computer storage media. By way of example, Fig. 29 illustrates a hard disk drive 2916 for reading from and writing to a non-removable, non-volatile magnetic media (not shown), a magnetic disk drive 2918 for reading from and writing to a removable, non-volatile magnetic disk 2920 (e.g., a "floppy disk"), and an optical disk drive 2922 for reading from and/or writing to a removable, non-volatile optical disk 2924 such as a CD-ROM, DVD-ROM, or other optical media. The hard disk drive 2916, magnetic disk drive 2918, and optical disk drive 2922 are each connected to the system bus 2908 by one or more data media interfaces 2926. Alternatively, the hard disk drive 2916, magnetic disk drive 2918, and optical disk drive 2922 can be connected to the system bus 2908 by one or more interfaces (not shown).

[00124] The disk drives and their associated computer-readable media provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for computer 2902. Although the example illustrates a hard disk 2916, a removable magnetic disk 2920, and a removable

optical disk 2924, it is to be appreciated that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes or other magnetic storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other optical storage, random access memories (RAM), read only memories (ROM), electrically erasable programmable read-only memory (EEPROM), and the like, can also be utilized to implement the exemplary computing system and environment.

[00125] Any number of program modules can be stored on the hard disk 2916, magnetic disk 2920, optical disk 2924, ROM 2912, and/or RAM 2910, including by way of example, an operating system 2926, one or more application programs 2928, other program modules 2930, and program data 2932. Each of such operating system 2926, one or more application programs 2928, other program modules 2930, and program data 2932 (or some combination thereof) may implement all or part of the resident components that support the distributed file system.

[00126] A user can enter commands and information into computer 2902 via input devices such as a keyboard 2934 and a pointing device 2936 (e.g., a "mouse"). Other input devices 2938 (not shown specifically) may include a microphone, joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and other input devices are connected to the processing unit 2904 via input/output interfaces 2940 that are coupled to the system bus 2908, but may be

connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB).

[00127] A monitor 2942 or other type of display device can also be connected to the system bus 2908 via an interface, such as a video adapter 2944. In addition to the monitor 2942, other output peripheral devices can include components such as speakers (not shown) and a printer 2946 which can be connected to computer 2902 via the input/output interfaces 2940.

[00128] Computer 2902 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computing device 2948. By way of example, the remote computing device 2948 can be a personal computer, portable computer, a server, a router, a network computer, a peer device or other common network node, game console, and the like. The remote computing device 2948 is illustrated as a portable computer that can include many or all of the elements and features described herein relative to computer 2902.

[00129] Logical connections between computer 2902 and the remote computer 2948 are depicted as a local area network (LAN) 2950 and a general wide area network (WAN) 2952. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

[00130] When implemented in a LAN networking environment, the computer 2902 is connected to a local network 2950 via a network interface or

adapter 2954. When implemented in a WAN networking environment, the computer 2902 typically includes a modem 2956 or other means for establishing communications over the wide network 2952. The modem 2956, which can be internal or external to computer 2902, can be connected to the system bus 2908 via the input/output interfaces 2940 or other appropriate mechanisms. It is to be appreciated that the illustrated network connections are exemplary and that other means of establishing communication link(s) between the computers 2902 and 2948 can be employed.

[00131] In a networked environment, such as that illustrated with computing environment 2900, program modules depicted relative to the computer 2902, or portions thereof, may be stored in a remote memory storage device. By way of example, remote application programs 2958 reside on a memory device of remote computer 2948. For purposes of illustration, application programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computing device 2902, and are executed by the data processor(s) of the computer.

[00132] Various modules and techniques may be described herein in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform

particular tasks or implement particular abstract data types. Typically, the functionality of the program modules may be combined or distributed as desired in various implementations.

[00133] An implementation of these modules and techniques may be stored on or transmitted across some form of computer-readable media. Computer-readable media can be any available media that can be accessed by a computer. By way of example, and not limitation, computer-readable media may comprise "computer storage media" and "communications media."

[00134] "Computer storage media" includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computer.

[00135] "Communication media" typically includes computer-readable instructions, data structures, program modules, or other data in a modulated data signal, such as carrier wave or other transport mechanism. Communication media also includes any information delivery media. The term "modulated data signal"

means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared, and other wireless media. Combinations of any of the above are also included within the scope of computer-readable media.

[00136] **EXEMPLARY APPLICATION PROGRAMS AND DATA**

[00137] Fig. 30 illustrates an exemplary block diagram that shows further details of the system memory 2906 of Fig. 29, including the application programs 2928 and the program data 2932 to present the pop-up light field. In this implementation, the application programs 2928 includes, for example, a layer pop-up module 3002 (e.g., 1504 of Fig. 15), a coherence matting module 3004 (e.g., to implement the method 700 of Fig. 7), a background construction module 3006 (e.g., to implement the method 2200 of Fig. 22), a foreground refinement module 3008 (e.g., 1510 of Fig. 15), an interpolation module 3010 (such as discussed with reference to the layered rendering algorithm), an alpha modeling module 3012 (e.g., to implement the stage 710 of Fig. 7), a rendering module 3014 (e.g., to implement the real-time rendering of pop-up field), and a boundary propagation module 3016 (e.g., to implement the stage 704 of Fig. 7).

[00138] The program data 2932 includes an original image(s) 3018, pop-up layer(s) 3020 (such as those discussed with reference to Figs. 4-5 and 16-20),

foreground layer(s) 3022 (such as 402 of Fig. 4), background layer(s) 3024 (such as 404 of Fig. 4), alpha values 3026 (such as those discussed with reference to Fig. 11), alpha-blend layer(s) 3028 (such as those discussed with respect to the layered rendering algorithm), rendered layer(s) 3030 (such as those discussed with respect to the layered rendering algorithm), intermediate image(s) 3032 (such as those used for Fig. 18), texture image(s) 3034 (such as those discussed with respect to the layered rendering algorithm), and other data 3036 (such as those shown in Fig. 30).

[00139] **LIGHT FIELD RENDERING ACCELERATION**

[00140] Light field rendering may be accelerated using graphics hardware. In a layered rendering algorithm, layers are alpha-blended using alpha values assigned in texture images, which means each layer is rendered onto the frame buffer in a single pass. One straightforward way is to copy the frame buffer into memory and composite them after rendering each layer.

[00141] Fig. 31 illustrates an exemplary single pass rendering method 3100 to provide improved speed. The method 3100 uses multi-texture mapping and programmable texture blending which is available on modern graphics hardware.

[00142] To blend all textures on a single triangle, a stage 3102 determines the texture-mapped triangles for each layer. Three different textures may be assigned to each triangle. A stage 3104 binds the textures to each triangle. In a stage 3106, three blending ratios $\{w_1,w_2,w_3\}$ are assigned on each triangle vertex as the

primary color {R,G,B}. The primary color may be smoothly interpolated on the triangle. Hence, the interpolated blending ratios $\{\tilde{v}_i\}$ are obtained by referring to the primary color at an arbitrary point on the triangle. Then, the texture images on the triangle can be blended (3108), for example, by using the blending equation programmed in the pixel shader in graphics hardware.

[00143] Accordingly, the layers can be composed by alpha-blending each triangle on the frame buffer when it is rendered because the triangles are arranged without overlap in a layer and each triangle is drawn in a single pass. In one implementation, OpenGL and its extensions for multi- texturing and per-pixel shading can be used.

[00144] **CONCLUSION**

[00145] Thus, although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.